Resource auto-scaling for SQL-like queries in the cloud based on parallel reinforcement learning

Mohamed Mehdi Kandi*, Shaoyi Yin and Abdelkader Hameurlain

IRIT Laboratory,
Paul Sabatier University,
Toulouse, France
Email: mohamed.kandi@irit.fr
Email: shaoyi.yin@irit.fr
Email: abdelkader.hameurlain@irit.fr
*Corresponding author

Abstract: Cloud computing is a technology that provides on-demand services in which the number of assigned resources can be automatically adjusted. A key challenge is how to choose the right number of resources so that the overall monetary cost is minimised. This problem, known as auto-scaling, was addressed in some existing works but most of them are dedicated to web applications. In these applications, it is assumed that the queries are atomic and each of them uses a single resource for a short period of time. However, this assumption cannot be considered for database applications. A query, in this case, contains many dependent and long tasks so several resources are required. We propose in this work an auto-scaling method based on reinforcement learning. The method is coupled with placement-scheduling. In the experimental section, we show the advantage of coupling the auto-scaling to the placement-scheduling by comparing our work to an existing auto-scaling method.

Keywords: cloud computing; auto-scaling; resource allocation; parallel reinforcement learning.

Reference to this paper should be made as follows: Kandi, M.M., Yin, S. and Hameurlain, A. (2019) 'Resource auto-scaling for SQL-like queries in the cloud based on parallel reinforcement learning', *Int. J. Grid and Utility Computing*, Vol. 10, No. 6, pp.654–671.

Biographical notes: Mohamed Mehdi Kandi is a PhD student at Paul Sabatier University, Toulouse, France. He received his Master's degree in Computer Science from Paris 6 University, France, in 2016. Now he works in the Pyramid team of IRIT Laboratory. His research interests include optimisation and elastic resource allocation for database applications in the cloud.

Shaoyi Yin conducted her PhD at the University of Versailles, France, under an INRIA doctoral contract and defended in June 2011. Since September 2012, she works at Paul Sabatier University, in the Pyramid team of IRIT Laboratory, as an associate professor. Her current research interests mainly include query optimisation in parallel and large-scale distributed environments, especially in the cloud environment.

Abdelkader Hameurlain is a full professor in Computer Science at Paul Sabatier University (IRIT Lab.) Toulouse, France. His current research interests are query optimisation in parallel and large-scale distributed environments, and mobile databases. He has been the general chair of the Int. Conf. on Database and Expert Systems Applications (DEXA02, DEXA11, DEXA17 and DEXA18). He is a Co-editor in Chief of the international journal *Transactions on Large-Scale Data- and Knowledge-Centered Systems* (LNCS, Springer).

1 Introduction

1.1 Context

Since it was launched in the 2000s, the cloud computing market continues to evolve. Today's cloud providers offer a variety of information technology services, from basic computing resources to complex applications that aim to meet the different tenant needs. Cloud services are based on a pricing model and Service Level Agreements (SLAs). The pricing model describes how services are billed. SLAs define the performance objectives that should be met by the provider and the measures to be taken in case of non-compliance with the objectives. If requirements are not respected, the provider pays penalties to the tenant.

Among cloud applications, we are interested in those concerning querying databases. Data processing has experienced in recent years the arrival of frameworks based on the storage and processing of partitioned data over a set of machines, mainly Hadoop ecosystem. Extensions that propose a syntax close to the classical SQL language (Hive) have been added to these frameworks in order to make them accessible to users who are accustomed to SQL. This syntax is called SQL-like. A submitted SQL-like query is transformed into many MapReduce jobs (Dean and Ghemawat, 2010), or a single job consisting of a Directed Acyclic Graph (DAG) (Saha et al., 2015), then assigned to available resources (Kllapi et al., 2011; Vavilapalli et al., 213). Cloud providers have quickly adapted to the new data processing frameworks by offering services like Amazon Elastic MapReduce and Microsoft Azure HDInsight.

1.2 Problem position

Maximising the profit is an important issue for cloud providers, but the existing data processing frameworks are not designed in a way that takes into account the pricing model and SLAs in the resource allocation process. Expenditures can be reduced if resource management handles these aspects. The number of assigned resources should increase if the load is high and decrease if the demand is low. If it is done in an automated way, this process is called auto-scaling.

The questions, in this case, are: when to scale and how much resources we should add or remove. Most current cloud providers use an intuitive approach based on thresholds for virtual machines (VMs) level auto-scaling and several related methods are proposed in literature (Khatua et al., 2010; Chieu et al., 2011; Simmons et al., 2011; Ghanbari et al., 2011; Han et al., 2012; Hasan et al., 2012). The main idea of this approach is to add (or remove) resources if a certain metric is above (or below) a predefined threshold. The thresholds are usually defined by humans in the provider or the tenant side.

The main drawback of the thresholds based approach is the fact that it requires a deep understanding of workload trends to choose good thresholds, which is not easy to achieve. This is why much scientific work was done to design human independent auto-scaling methods based mainly on reinforcement learning (Dutreilh et al., 2011; Rao et al., 2011). The learner is an agent that makes successive actions in an environment and receives a reward for each taken action. After a set of random trials, he should learn how to take good actions which avoid human intervention.

In practice, there is no general reinforcement learning solution that provides an efficient scaling for all possible applications and architectures. Each solution must consider the environment, the cost objective and constraints of a specific application and we notice that the most existing methods focus on web querying. A query in these applications can be seen as an atomic task assigned to a single resource for a short time period. These methods are not suitable for database querying applications. Indeed, the structure of a SQL-like query is considered complex compared to other common applications. The physical execution plan of the query contains many tasks. Some of them are parallel, others are linked by a producer-consumer relation. So parallelism and communication must be considered.

Existing works that use reinforcement learning for auto-scaling are based on the information about the present to define the state description and cost function. This may be sufficient for applications in which queries consist of one or few tasks with a short execution time but databases queries consist of several dependent tasks with relatively long execution time. Information about the present may generate under or over scaling. Indeed, a query launched at the moment t may end at the moment $t+\Delta t$ where Δt is not negligible. This is why it is also important to have some knowledge about the future during the scaling decision.

Moreover, in a database query, if the consumer is not launched at the moment the data is generated by the producer then the intermediate data can be stored on disk. The storage in the cloud has a significant monetary cost.

1.3 Contribution

In this work, we propose an auto-scaling method based on reinforcement learning for database querying in the cloud. In order to satisfy scalability and accelerate learning, the method applies a parallel variant of reinforcement learning (Kretchmar, 2002; Grounds and Kudenko, 2008; Barrett et al., 2013). In this variant, the number of agents is greater than one. Agents work independently but share their experience to learn quickly. This variant is recommended for database querying given the large number of queries and resources to manage.

The main originality of our work is the fact that we estimate the resource availability and penalties of a future time window in order to provide more accurate state description and reward function in the reinforcement learning process. These estimations are injected into the reinforcement learning algorithm to improve agents learning ability. The computation of the estimations is based on the placement and scheduling plan. We refer to the overall process (auto-scaling, placement, and scheduling) as elastic resource allocation.

Another originality of our work is the fact that we take into account the storage of intermediate data in the reward function of the reinforcement learning algorithm. Storage is an aspect that characterises database applications. The intermediate data can be temporarily stored on disk if the consumer is not yet available and storage in the cloud has a significant monetary cost.

The rest of this paper is organised as follows: Section 2 explains the context and an overview of the elastic resource allocation process. We give a detailed explanation of the proposed auto-scaling method in Section 3. Section 4 presents the experimental environment and the results. An overview of the existing work on auto-scaling in the cloud is provided in Section 5. We conclude in Section 6.

2 Elastic resource allocation process: an overview

In this section, we give an overview of the query compilation context (2.1). Then, we explain the idea of the elastic resource allocation process that we propose and the intuition behind the placement-scheduling phases (2.2). Finally, we present the performance model (2.3). More details about placement and scheduling can be found in Appendix A and B.

2.1 SQL-like query compilation

Figure 1 shows the SQL-like query compilation process. Tenants submit SQL-like queries using the client interface. First, the lexical and syntactic analyser verifies that the query is syntactically correct. Then, the logical optimisation applies some transformation rules in order to reduce the volume of manipulated data during the query execution. The physical optimisation chooses the join algorithms, defines the join order and generates the execution plan. Finally, the paralleliser defines the degree of parallelism and generates a Directed Acyclic Graph (DAG) of tasks.

Older versions of parallel data processing framework represent this graph by a set of dependent MapReduce jobs (Figure 2a). MapReduce (Dean and Ghemawat, 2008) is recognised as an efficient parallel programming model. Indeed, it allows performing computation on massive data partitioned on a large number of nodes. However, the classical MapReduce model has some drawbacks. The graph is represented by a set of dependent jobs. Between two successive jobs, there is a write operation, then a read operation on the Distributed File System (DFS). In addition, the pipeline is not planned and each job must contain a Map and a Reduce phase even if one of them is not necessary.

Figure 1 SQL-like query compilation process

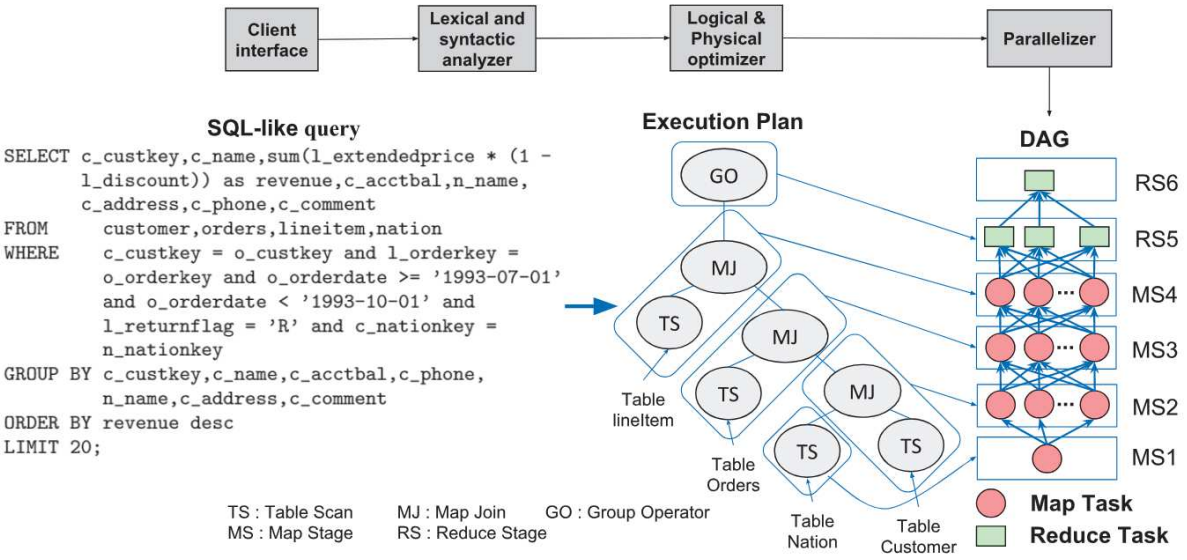
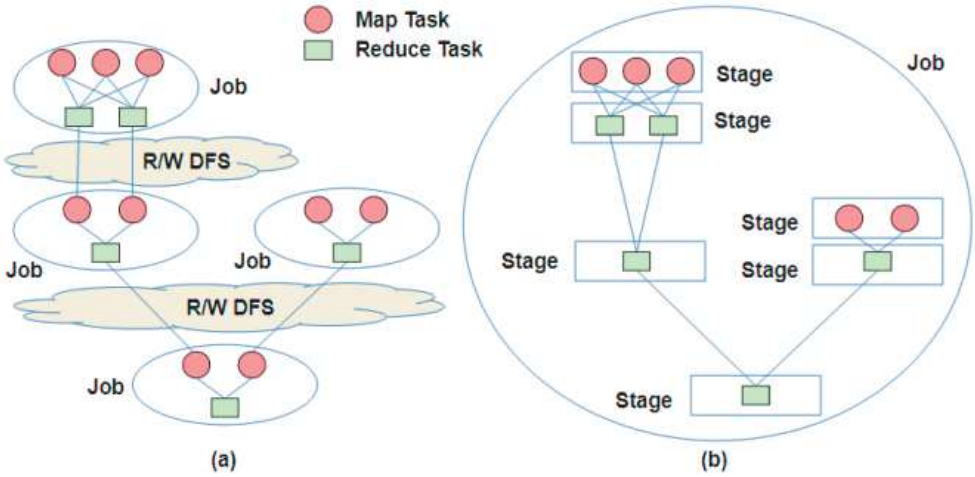


Figure 2 Comparison of the graph structure in (a) the multi-job model and (b) the single-job model



Recently, another model has been proposed and already integrated into existing tools, mainly Hive/Tez (Saha et al., 2015) and SparkSQL (Armbrust et al., 2015).[1] It consists of using a single job for the entire query (Figure 2b). The internal structure is flexible. Indeed, the query is represented by a set of stages. A stage can perform either a Map or Reduce function. Each stage has many parallel tasks. The task does the processing on part of the data. The pipeline is allowed between the linked stages and read/write of intermediate data in DFS is avoided.

In this paper, we refer to the classical MapReduce as the multi-job model and the new representation as the single-job model. In Hive/Tez, the query in Figure 1 contains three Map Joins (MJ) and a Group Operator (GO). It is represented by a DAG with four Map stages and two Reduce stages.

2.2 *Elastic resource allocation process*

The cloud service infrastructure consists of a set of physical machines (Figure 4). A hypervisor, whose role is to manage VMs, is installed on each physical machine. A VM contains a set of logical resources. A logical resource is an abstract representation of a certain amount of reserved CPU, memory and storage. The system receives DAGs of submitted queries. The capacity manager adjusts the number of logical resources reserved for the service (auto-scaling). The global resource allocation manager performs task placement and scheduling given the available logical resources. Each elementary task is assigned to a particular logical resource for a given time period.

Figure 3 illustrates the elastic allocation process that we propose. This process is a sequence of four steps: (1) auto-scaling, (2) choice of method, (3) placement and (4) scheduling. These steps are performed one after another. A complete execution of the four steps is called an iteration. A new iteration is performed in each fixed unit of time dt. In a given iteration launched at the moment t, the system handles the DAGs received between $(t - dt)$ and t.

Auto-scaling is the process of determining the number of logical resources required to meet the load with minimum cost. We give more details about the proposed auto-scaling method in Section 3. Following the auto-scaling, the agent chooses the best method for placement and scheduling of tasks based on graph sizes and the number of assigned resources. The placement consists of choosing a logical resource for each task (Figure 5a). Scheduling allows choosing the time window assigned to each task (Figure 5b). At the end of each iteration of this process, the final placement-scheduling plan, as well as the estimated costs of penalties and storage of the intermediate data are injected into the capacity manager in order to be used for the scaling of the next iteration.

Figure 3 Parallel elastic allocation process

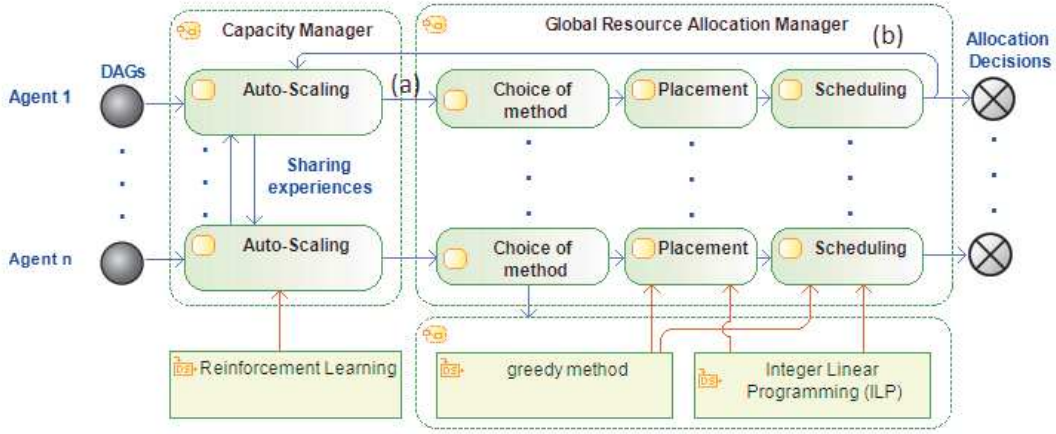


Figure 4 Cloud service architecture

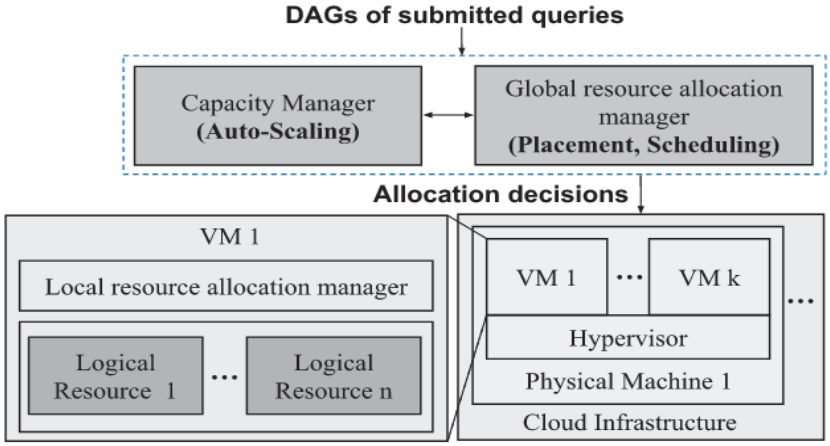
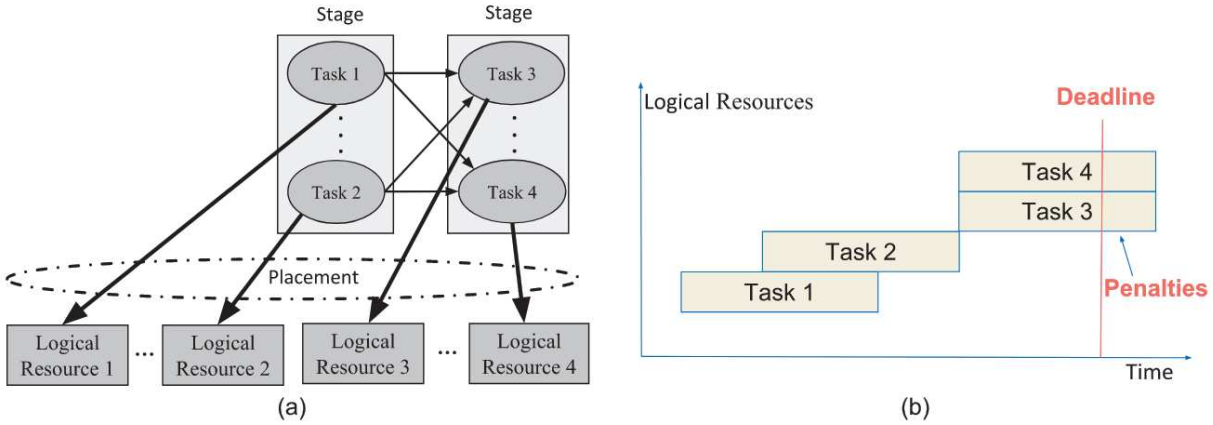


Figure 5 Intuition behind the (a) placement and (b) scheduling



There are n agents that work independently. In our architecture, only one database querying application is considered. Each agent is responsible for multiple queries. There are two benefits of using n agents working in parallel: (1) experiences can be shared between agents which accelerate the learning, (2) the elastic allocation process is faster when there are many agents. To explain the second point we consider the following example. We assume that at the moment t, there are 50 allocated logical resources and 20 queries arriving in the system. Consider the first case where we have only one agent, the latter must do the placement-scheduling concerning 20 queries on 50 logical resources. Let's consider a second case where we have five parallel agents. In this case, each agent must do the placement-scheduling concerning four queries (20/5) on approximately ten logical resources (50/5). The problem is less complex in the second case since there are fewer queries and logical resources to manage by an agent, so placement-scheduling decision algorithms are faster.

Our auto-scaling solution is dependent on placement and scheduling. These latter are optimisation problems. In the field of optimisation, there are two main classes of methods: approximate and exact methods. Our elastic resource allocation process supports the approximate method proposed in Kllapi et al. (2011) as well as an exact method based on Integer Linear Programming (ILP) that is part of our research (Kandi et al., 2018). The ILP formulation is presented in Appendices A and B. A choice of the approach is made first based on the complexity of the queries received and the number of available resources.

2.3 Performance model

Since the elastic allocation process is performed before running queries, it is essential to estimate the duration of tasks. This estimation is based on the size of the data, the number of tuples, the selectivity of the operators and some system parameters (Table 1). The formula applied to estimate task duration depends on the operators executed by the task. For example, let's consider a stage Si that performs a selection on a relation $R1$ then a Map Join between the result of the selection and a relation $R2$. The duration (ET) of an elementary task of the stage Si is estimated as follows (Yin et al., 2018):

$$ET(Si) = sum($$

$$(|R1|*|S1|/pd_i)/db + dl;\ //read\ the\ data\ from\ disk$$

$$(|R1|*|S1|/pd_i)*(ipc/cpu);\ //execute\ de\ select\ operator$$

$$(|\sigma(R1)|*|S1|/pd_i)*(iph/cpu);\ //execute\ de\ prob\ operator$$

$$(|R2|*|S2|/pd_i)/db + dl;\ //read\ the\ data\ from\ disk$$

$$(|R2|*|S2|/pd_i)*(iph/cpu);\ //execute\ de\ build\ operator$$

$$(|R12|*|S12|/pd_i)*(iph/cpu);\ //prepare\ tuples\ partioning$$

$$(|R12|*|S12|*/(pd_i * pd_{i+1})/nb + nd)*max(pd_i, pd_{i+1})$$

$$//transmit\ the\ result\ to\ the\ next\ stage)$$

Table 1 Cost model parameters (Yin et al., 2018)

Parameter	*Signification*
$\|Rx\|$	number of tuples in Rx
$\|Sx\|$	size of a tuple in Rx (Bytes)
$\|Rxy\|$	number of tuples in $Rx \triangleright\triangleleft Ry$
$\|Sxy\|$	size of a tuple in $Rx \triangleright\triangleleft Ry$ (Bytes)
σ	selectivity
pd_i	number of tasks in the stage Si
db	disk I/O bandwidth
dl	disk latency
cpu	CPU processing speed
nb	network bandwidth
nd	network delay
ipc	number of instructions for comparing two bytes
iph	number of instructions for hashing a byte

3 Auto-scaling method

The goal of auto-scaling is to add or remove resources in order to respond to load variation. The proposed auto-

scaling method is based on parallel reinforcement learning. We explain in this section the reinforcement learning principle (3.1), then the modelling of the learning solution for the auto-scaling problem (3.2).

3.1 Reinforcement learning (RL) principle

The basic idea of reinforcement learning is that we have an agent who makes decisions in a complex environment. At each moment, the agent knows the current state of the system and takes an action that allows moving to another state. When the agent takes an action, the environment provides him a reward (or a penalty) – Figure 6. After a set of iterations, the agent should learn the sequence of actions that maximise the total reward (or minimise the total penalty). The operation of such a system is modeled by a Markov decision process (MDP). An MDP is a mathematical model represented by a state space (S), an action space (A), probabilities of transitions between states and rewards (R). At the moment t, the agent is in a state s_t, then he chooses an action $a_t \in A(s_t)$, such that $A(s_t)$ is the set of possible actions when the system is in the state s_t. Following the action, the agent receives a reward $r_{t+1} \in R$ and then moves to a state s_{t+1}. The value of the reward and the next state follow probability distributions: $p(r_{t+1} / s_t, a_t)$, $p(s_{t+1} / s_t, a_t)$. The goal of the agent is to maximise the cumulative reward (or minimise the cumulative penalty) in the long run. A measure $Q(s_t, a_t)$ is associated with each state (s_t) action (a_t) pair:

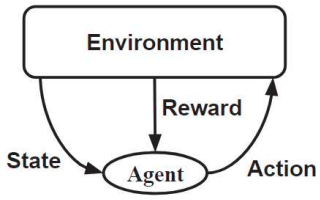
$$Q(s_t, a_t) = E(r_{t+1} / s_t, a_t) + \gamma * \sum_{s_{t+1}} P(s_{t+1} / s_t, a_t) * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (1)$$

where $E(r_{t+1} / s_t, a_t)$ is the expected value of $r_{t+1} / s_t, a_t$. $P(s_{t+1} / s_t, a_t)$ is the probability to move to the state s_{t+1} knowing that we are in the state s_t and we made the action a_t. $0 \le \gamma < 1$ is a discount factor helping the convergence of Q. The measure $Q(s_t, a_t)$ quantifies the possible actions. The best action is the one with the highest value of Q (or the lowest if r is seen as a penalty). When the MDP structure is well known, the values of Q can be computed with the value iteration method (Alpaydin, 2014). However, usually the structure of the system and its behaviour are not known in advance, and in this case, reinforcement learning algorithms, such as Q-learning (described in Alpaydin, 2014 and Watkins, 1989) is more appropriate. There are several variants of Q-learning. A variant called Sarsa is presented in Algorithm 1. The algorithm uses learning to find an estimated value ($\hat{Q}$). $\hat{Q}$ is initialised with an arbitrary value (line 1) then improved in each iteration as the agent explores the state space (line 13). The strategy to choose the next action should make a trade-off between exploring the state space and maximising the reward. For this, the agent chooses an action randomly with a probability ε (lines 8–9) and chooses the action with the highest $\hat{Q}(s,a)$ with the probability $1-\varepsilon$ (lines 10–11). The parameters η, γ and the function ε are defined in advance.

Algorithm 1: Q-Learning Sarsa (Alpaydin, 2014)

1: initialise all $\hat{Q}(s,a)$ arbitrarily
2: initialise s
3: choose an action $a \in A(s)$ randomly
4: $iteration \leftarrow 1$
5: **repeat**
6: observe s' (the new state) and $R(s,a)$ (the reward)
7: generate a random number "*rand*" between 0 and 1
8: **if** $rand < \varepsilon(iteration)$ **then**
9: choose an action $a' \in A(s')$ randomly
10: **else**
11: choose an action $a' \in A(s')$ with the highest $\hat{Q}$
12: **end if**
13: Update $\hat{Q}(s,a)$:
$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \eta * (R(s,a) + \gamma * \hat{Q}(s',a') - \hat{Q}(s,a))$
14: $s \leftarrow s'$
15: $a \leftarrow a'$
16: $iteration \leftarrow iteration + 1$
17: wait for the next iteration
18: **until** the system stops

Figure 6 Interaction between the agent and the environment



3.2 Modelling the learning solution for auto-scaling

We apply reinforcement learning to ensure that the cloud service described earlier has the correct number of logical resources. We define the state as a triplet: (1) the number of logical resources allocated for each type, (2) the current time of the day, and (3) the resource availability level.

Formally a state $s = (n, h, d)$. With $n = (n_1, n_2, ...n_{card(\mathcal{C})})$, $\mathcal{C}$ is the set of resources types, each type is characterised by its memory capacity and monetary cost, n_c is the number of assigned type c resources, h is a discrete representation of the current time of the day, $0 \le d \le 1$ define the resource availability level.

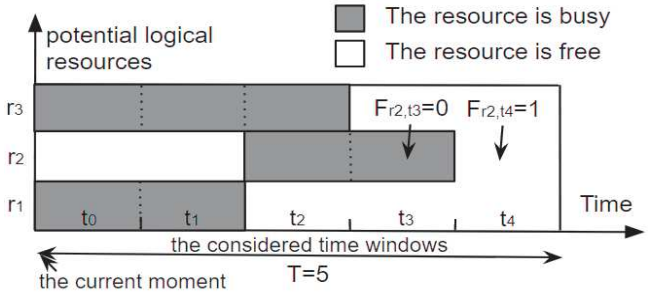
Unlike web applications, the duration of a database query can be long. The resource availability level in our method is not just based on the current moment (this is the case for most of the work of state-of-the-art work) but also on a future time window calculated from the resource

placement-scheduling plan. The availability level d is computed with the formula (2). With $\mathcal{P}(iter)$ is the set of potential logical resources in the current iteration 'iter', T is the number of considered future time windows, $F_{r,t}$ indicate whether the logical resource $r \in \mathcal{P}(iter)$ is free at the moment t ($F_{r,t}=1$) or busy ($F_{r,t}=0$) according to the placement-scheduling plan ($t < T$).

$$d = round\left(\frac{\sum_{r\in\mathcal{P}(iter)}\sum_{t<T}(1-F_{r,t})}{T * card(\mathcal{P}(iter))}, nbDigits\right) \quad (2)$$

In order to explain the intuition behind formula (2), we consider the example of Figure 7. The numerator of the formula represents the surface of the grey area (the resources are busy). The denominator represents the surface of the grey area plus the surface of the white area (the resources are busy + they are free). So d is a number between 0 and 1, with d is close to 0 (resp. close to 1) when there are many available resources (resp. many busy resources) in the considered future time window T. $nbDigits$ is the number of digits from the decimal point when we apply the *round* function. The *round* function is necessary to get a state space with a finite number of states.

Figure 7 Placement-scheduling plan example



Possible actions are to keep the same number (0), add (+1) or release (−1) a resource. An action $a = (a(1), a(2), .. a(card(\mathcal{C})))$ with $a(c) \in \{-1;0;1\} \forall c \in \mathcal{C}$ and if $\exists c_1 \in \mathcal{C}$, $a(c_1) = 1 \vee a(c_1) = -1 \Rightarrow a(c_2) = 0 \quad \forall c_2 \neq c_1$ (i.e. we don't add or remove more than one resource in a single iteration).

The reward following an action a includes: (1) the cost of using physical resources C_{ress}, (2) the cost of assigning and releasing resources C_{ajust}, (3) the costs of penalties paid by the provider in case of SLA violation C_{pen} and (4) the cost of using storage C_{stor}:

$$R(s,a) = C_{ress}(s) + C_{ajust}(a) + C_{pen}(s) + C_{stor}(s) \quad (3)$$

The cost of using resources includes the processor cost C_{proc} and memory C_{mem}:

$$C_{ress}(s) = C_{proc}(s) + C_{mem}(s) \quad (4)$$

The processor (resp. memory) cost is calculated as follows:

$$C_{proc}(s) = W_{proc} * \sum_{c\in\mathcal{C}} s.n_c \quad (5)$$

$$C_{mem}(s) = W_{mem} * \sum_{c\in\mathcal{C}} C_m(c) * s.n_c \quad (6)$$

W_{proc} is the processor cost weight, W_{mem} is the memory cost weight, $s.n_c$ is the number of logical resources of type c when the system is in state s, $C_m(c)$ is the available memory in the type c resources.

The adjustment cost depends on the cost of adding resources C_{asg} and the cost of removing resources C_{rel} ($1_{condition} = 1$ if the condition is true, $= 0$ otherwise):

$$C_{ajust}(a) = \sum_{c\in\mathcal{C}} \left(C_{asg}(c) * 1_{a(c)>0} + C_{rel}(c) * 1_{a(c)<0} \right) \quad (7)$$

Penalties (C_{pen}) and storage (C_{stor}) costs depend on task scheduling (Appendix B). Based on the objective function of the ILP scheduling model and assuming that the optimal solution of this ILP is denoted $\{y^*, w^*, \beta^*\}$, we have (the notation is described in Table A1):

$$C_{pen}(s) = \sum_{i\in\mathcal{F}} \sum_{D_i - T_i < t < T} W_i * \beta^*_{i,t} \quad (8)$$

$$C_{stor}(s) = W_s * \sum_{i\in S} \sum_{m\in T_i} \sum_{t<T} q_i * w^*_{i,m,t} \quad (9)$$

The goal is to minimise the cumulative reward (R) in long run. The best action is the one with the lowest value of Q (formula (1)). In our problem, the behaviour of the MDP is not known in advance so, as mentioned in the end on Subsection 3.1, an algorithm such as Q-learning can be used to learn an estimated value $\hat{Q}$. The value of $\hat{Q}$ is improved in each iteration based on the observed reward and the state description.

In reinforcement learning, the number of times a state is visited determines the quality of the decision. Indeed, more visits imply a better experience. It is therefore interesting to adopt methods that accelerate the evolution of the learning agent's experience. We propose to adopt parallel reinforcement learning (Kretchmar, 2002) to accelerate learning. The system has a set of parallel agents who share their experiences. Each agent i makes his decisions using a measure $\hat{Q}^i$ consisting of his own local experience $\hat{Q}^i_l$ and the global experience $\hat{Q}^i_g$ that the other agents shared with him.

Algorithm 2 illustrates the parallel reinforcement learning applied to our problem and executed by an agent i among a set of agents. In this algorithm: $k^i(s,a)$ is the number of times the action a was taken following a visit to the state s. The agent can give more importance to his own experience and so, in this case, W_l and W_g are chosen such that $W_l > W_g$. Algorithm 2 is executed by the capacity manager (represented in Figures 3 and 4).

Algorithm 2: Parallel reinforcement learning (Agent i)

1: initialise all $\hat{Q}_l^i(s,a)$ in an arbitrary way

2: initialise all $k^i(s',a')$ to 1

3: **for** each state $s \in S$ **do**

4: $C_{proc}(s) \leftarrow W_{proc} * \sum_{c \in \mathcal{C}} s.n_c$

5: $C_{mem}(s) \leftarrow W_{mem} * \sum_{c \in \mathcal{C}} C_m(j) * s.n_c$

6: $C_{ress}(s) \leftarrow C_{proc}(s) + C_{mem}(s)$

7: **for** each action $a \in A(s)$ **do**

8: $C_{ajust}(a) \leftarrow \sum_{c \in \mathcal{C}} C_{asg}(c) * 1_{a(c)>0} + \sum_{c \in \mathcal{C}} C_{rel}(c) * 1_{a(c)<0}$

9: **end for**

10: **end for**

11: initialise s

12: choose an action $a \in A(s)$ randomly

13: **repeat**

14: receive DAGs of the new submitted queries

15: execute the action a (scaling)

16: notify the global resource allocation manager that the scaling is done (Figure 3, arrow (a)), the placement and scheduling of new queries is therefore launched,

17: get the estimated $C_{stor}(s)$ and $C_{pen}(s)$ from the resource allocation manager (Figure 3, arrow (b)),

18: $R(s,a) \leftarrow C_{ress}(s) + C_{ajust}(a) + C_{pen}(s) + C_{stor}(s)$

19: observe s' (the new state)

20: **for** each action $a' \in A(s')$ **do**

21: $K \leftarrow \sum_{j \in Agents, j \neq i} k^j(s',a')$

22: $\hat{Q}_g^i(s',a') \leftarrow \sum_{j \in Agents, j \neq i} k^j(s',a') * \hat{Q}_l^j(s',a') K$

23: $\hat{Q}^i(s',a') \leftarrow$

$W_l * k^i(s',a') * \hat{Q}_l^i(s',a') + W_g * K * (nbAgents - 1) * \hat{Q}_g^i(s',a') W_l * k^i(s',a') + W_g * K * (nbAgents - 1)$

24: **end for**

25: generate a random number '*rand*' between 0 and 1

26: **if** $rand < \varepsilon(iteration)$ **then**

27: choose an action $a' \in A(s')$ randomly

28: **else**

29: choose an action $a' \in A(s')$ with the highest $\hat{Q}^i$

30: **end if**

31: update $\hat{Q}_l^i(s,a)$:

32: $\hat{Q}_l^i(s,a) \leftarrow \hat{Q}_l^i(s,a) + \eta * (R(s,a) + \gamma * \hat{Q}_l^i(s',a') - \hat{Q}_l^i(s,a))$

33: $k^i(s,a) \leftarrow k^i(s,a) + 1$

34: share $\hat{Q}_l^i(s,a)$ and $k^i(s,a)$ with the others agents

35: $s \leftarrow s'$

36: $a \leftarrow a'$

37: $iteration \leftarrow iteration + 1$

38: wait for the next iteration

39: **until** the system stops

4 Experimental results

We present an experimental evaluation of the auto-scaling method. In Subsection 4.1, we give a general overview of the performed simulation. In Subsection 4.2, we compare our method that uses placement-scheduling plan to estimate the future resource availability, penalties and storage usage with an auto-scaling method that works independently of placement-scheduling. We illustrate the advantage of our method in terms of monetary cost. In Subsection 4.3, we show the impact of experience sharing and how the method scale. We assume that some agents exploit the shared experience and others not and evaluate the monetary cost and allocation duration. In Subsection 4.4, we compare two variants of the algorithm: the standard Q-learning and Sarsa. The goal is to justify the choice of the Sarsa variant for our solution

4.1 Simulation setup

We evaluate the proposed solution by simulation. The queries are retrieved from the TPC-H benchmark then tested on Hive (a version based on Tez) in order to define: (1) the structure of DAGs, (2) the number of parallel tasks per stage and (3) the estimated size of the intermediate data. ILP models are solved with GLPK. We assume that each agent handles 10 simulated VMs that contain eight logical resources each. We have two types of VMs: type 1 VMs (composed of eight logical resources with 256MB of memory each, price: 0.25$/hour) and type 2 VMs (composed of eightlogical resources with 512MB of memory each: 0.5$/hour). For a given query, the provider pays a penalty of 0.1$ for each minute after the deadline specified in the SLAs. We assume also that the cost of adjusting resources (add or remove a resource) is 0.1$.

Our simulation was performed on a computing node with four AMD processors and 512 GB of RAM. In all experiments, the global monetary cost is simulated. The global monetary cost at a given iteration is the sum of the following costs: 1) used logical resources, 2) penalties caused by violation of deadlines, 3) adjusting resources and 4) storage of data of consumers are not yet ready. The global monetary computation is based on formula (3) – Section 3.2.

In the simulation, we run Algorithm 2 for a large number of iterations (up to 70000 iterations). An iteration is defined as a complete execution of lines from 14 to 38 of the algorithm. At the beginning of each iteration, we assume the arrival of a number of queries represented by their DAG (line 14 of the algorithm). The number of received queries follows a random distribution that depends on the time of day. The number of queries received on the day is more important than the number received at night. The maximum number of queries considered is 540 queries/hour. All these queries have a DAG that follows the TPC-H benchmark. It is important to mention that a query started at the iteration i may end later (i.e. at iteration $i+j$, $j>0$). In each iteration, a scaling decision is made (line 25 to 30 of the algorithm).

We set $\eta = 0.5$ and $\alpha = 0.8$. We recall that Algorithm 2 uses the parameter ε to make the compromise between the choice of the right decisions and the exploration of the state space. In order to test different scenarios we propose three policies:

- Policy 1 (three steps):

$$\varepsilon(iteration) = \begin{cases} 0.999, \ when \\ 0 \le iteration < endStep1 \\ 0.4, \ when \\ endStep1 \le iteration < endStep2 \\ 0.2, \ when \\ endStep2 \le iteration < endOfSimul \end{cases}$$

- Policy 2 (two steps):

$$\varepsilon(iteration) = \begin{cases} 0.999, \ when \\ 0 \le iteration < endStep1 \\ 0.3, \ when \\ endStep1 \le iteration < endOfSimul \end{cases}$$

- Policy 3 (one step):

$$\varepsilon(iteration) = \begin{cases} 0.1, & when \ 0 \le iteration < endOfSimul \end{cases}$$

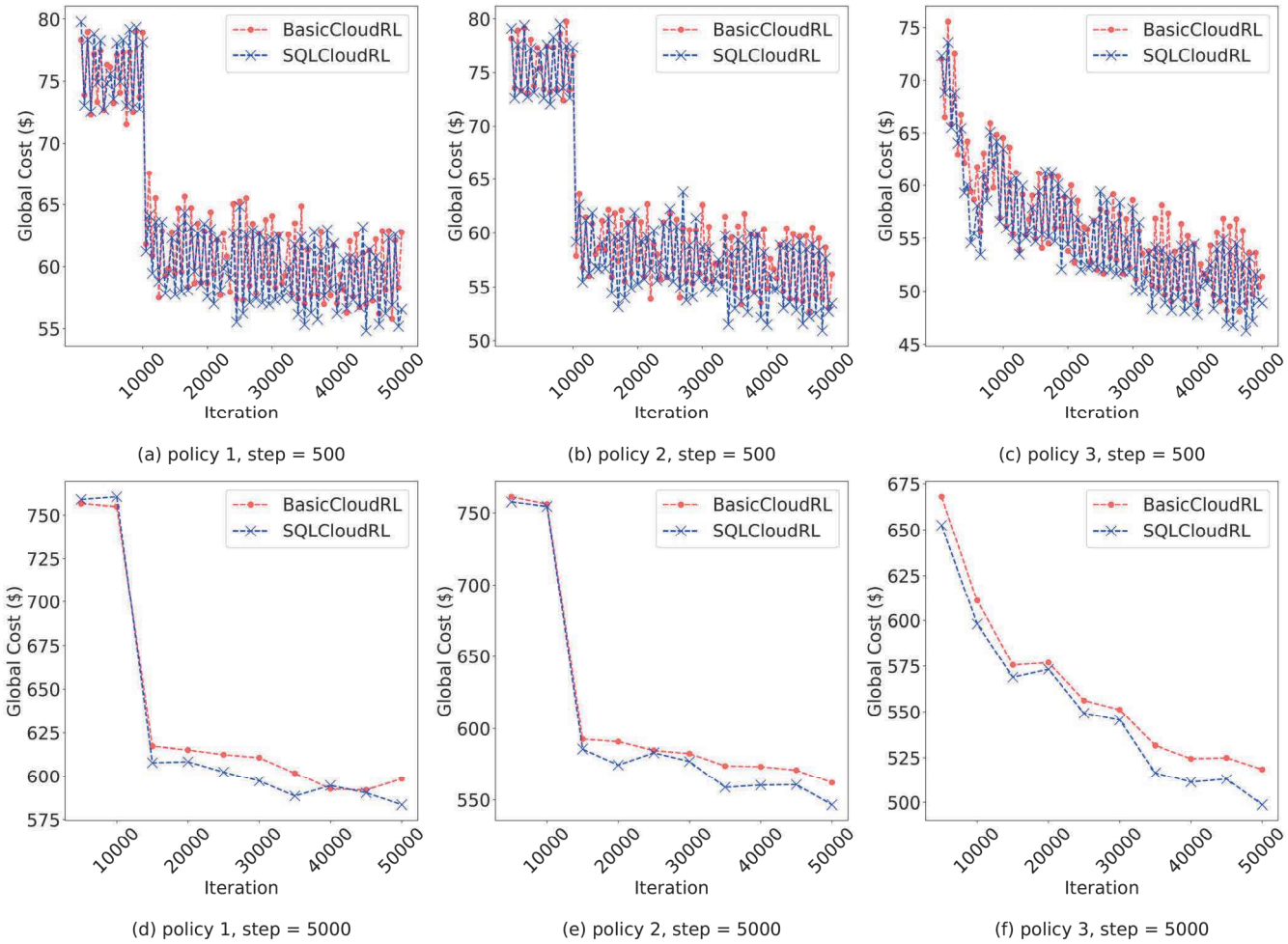
4.2 *Experiment 1: comparison of our method and basic reinforcement learning method (SQLCloudRL vs. BasicCloudRL)*

In this section, we compare our work with an auto-scaling method from the literature. We chose to not compare our solution with threshold-based methods. The difference between the reinforcement learning approach and the threshold approach is discussed in the related work section. The advantage of reinforcement learning is the fact that it is independent of human intervention. If we compare our work with a method based on thresholds we will have to manually set the values of the thresholds, so performing a fair comparison is not obvious.

We have therefore compared our work with a method of literature also based on reinforcement learning. The baseline is similar to the solution presented in Dutreilh et al. (2011). The particularity of our work is the dependence between the auto-scaling and the placement-scheduling. Indeed, as explained in the Subsection 3.2, the output of the placement-scheduling is used to give a more precise representation of the states of the MDP and the reward function. The baseline, contrariwise, uses the Q-learning algorithm but assumes that auto-scaling is performed independently of the outputs of the placement-scheduling. In the following, we call our proposal SQLCloudRL (SQL-like Queries Cloud Reinforcement Learning). The baseline is named BasicCloudRL (Basic Cloud Reinforcement Learning).

In Figure 8 the global cost ($) is cumulative over an iteration interval. The value of 'step' indicates the length of this interval. Three agents are considered in each test and the given values in y-axis correspond to the average monetary cost per time unit of the three agents. At iteration 1, each agent has no knowledge of the environment so the monetary cost is high at the beginning. The agent explores the environment, stores its experiences and exploits it in the decision making which makes the monetary cost decreases.

Figure 8 Evolution of monetary cost over time (SQLCloudRL vs. BasicCloudRL)



On the one hand, the scaling decisions for policy 1 and 2 are almost random at the beginning ($\varepsilon(iteration) = 0.999$ when $0 \leq iteration < 10000$). The global costs, therefore, remain stable and high in this period of time. The random decisions allow the agent to explore more possibilities so learning is fast. As soon as we change the value of ε at $iteration = 1000$, the agent starts using the stored experience to make the scaling decisions so the costs drop sharply. Policy 3, on the other hand, uses a progressive approach. The agent makes a trade-off between exploration and optimisation from the beginning ($\varepsilon(iteration) = 0.1 \quad \forall iteration$). The cost, therefore, decreases in a less brutal way than policies 1 and 2.

The same figure shows the advantage of our formulation compared to the existing auto-scaling method in terms of monetary cost. The benefit of our proposal is due to the consideration of estimate the future resource availability level and penalties from the previous placement-scheduling plan. The formulation of the baseline can be useful for short queries (web applications) but not sufficient for long queries of database querying applications. For long queries, considering a future estimate allows a more precise definition of a state of the MDP. The gain in terms of monetary cost becomes very significant in a real cloud with tens or even hundreds of agents and after a large number of iterations.

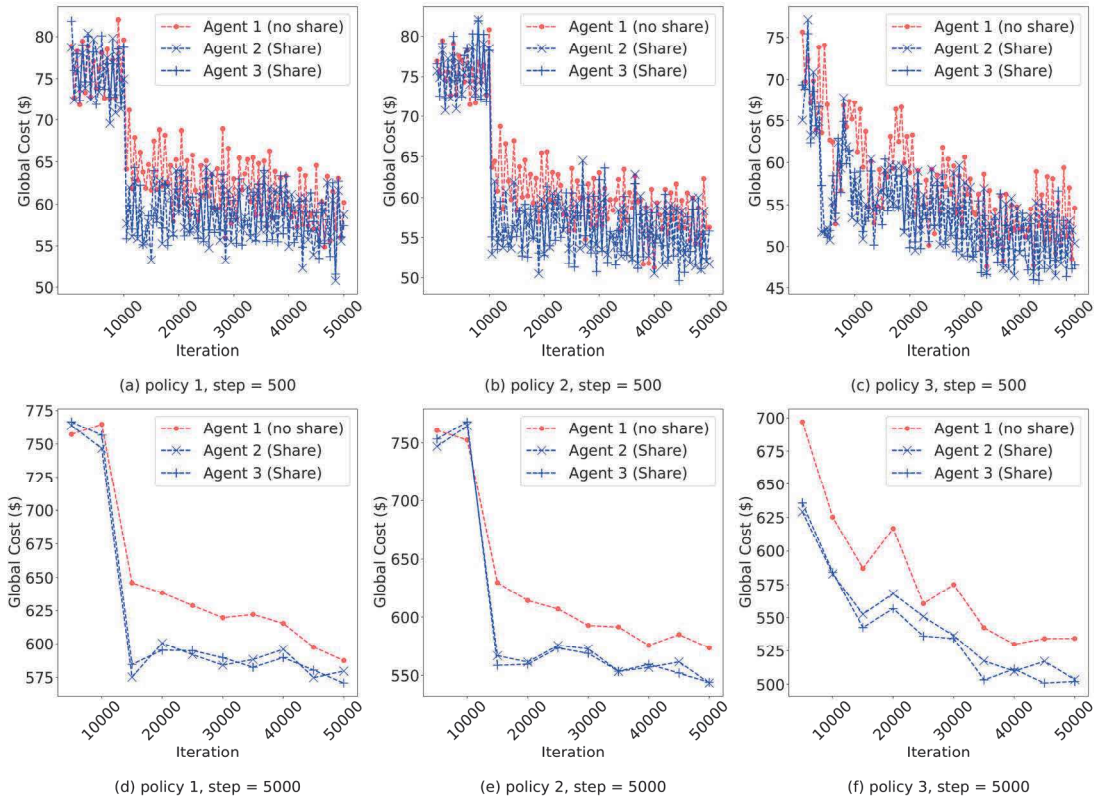
4.3 Experiment 2: impact of experience sharing (no share vs. share)

We consider three agents who work in parallel and share their experiences. Agents 2 and 3 use the shared experience (share) while agent 1 uses only its own experience (no share). Table 2 and Figure 9 show the results. We note the same remarks as the previous experiment 1 regarding the overall trend of the curves. In addition, the cost values for the agents who benefit from the shared experience (agents 2 and 3) are lower than those who do not (agent 1). Indeed, sharing experience allows agents 2 and 3 to learn faster than agent 1.

Table 2 Average monetary cost ($) per iteration for Q-learning without (agent 1) and with (agents 2 and 3) experience sharing (phase 1: $0 \leq iteration < 10000$, phase 2: $10000 \leq iteration < 20000$, phase 3: $2000 \leq iteration$

	policy 1		
	phase 1	*phase 2*	*phase 3*
No share – cost	0.1522	0.1285	0.1224
Share – cost	0.1517	0.1178	0.1170
Gain		5.4%	
	policy 2		
	phase 1	*phase 2*	*phase 3*
No share – cost	0.1513	0.1245	0.1175
Share – cost	0.1517	0.1123	0.1119
Gain		6.1%	
	policy 3		
	phase 1	*phase 2*	*phase 3*
No share – cost	0.1322	0.1204	0.1092
Share – cost	0.1216	0.1110	0.1037
Gain	5.8%		

Figure 9 Evolution of monetary cost over time (no share vs. share)



Then we vary the number of agents and observe the evolution of the monetary cost and allocation duration: scaling+placement+scheduling (Figure 11). On the one hand, the monetary cost decreases with the increase in the number of agents. Indeed, more there are agents sharing their experience, faster is the learning and therefore the auto-scaling is done in a more efficient way. On the other hand, the increase in the number of agents implies more messages exchanged and therefore more allocation duration. To limit the allocation duration when the number of agents is very large, it is possible to group the agents into clusters such that each agent shares its experience only with the agents of his cluster. Another solution is not to exchange at each iteration but rather after a certain number of iterations.

4.4 Experiment 3: comparison of the standard Q-learning algorithm and the Sarsa variant (standard Q-learning vs. Sarsa)

We show in this experiment why we chose the Sarsa variant as a learning algorithm. The difference between the standard Q-learning algorithm and Sarsa is the way to update the values of $\hat{Q}(s,a)$. Standard Q-learning uses the best action a' of the next state s' to update $\hat{Q}(s,a)$ while Sarsa first chooses an action a' using the current policy and then returns to update $\hat{Q}(s,a)$. The two algorithms combine the exploration and the optimisation in the decision making but the standard Q-learning considers that the agent takes always the optimal policy when updating the values of $\hat{Q}$ whereas Sarsa considers the fact that actual policy combines the exploration and the optimisation (i.e. Sarsa allows the agent to learn that some of his decisions are random). The stored experience in Sarsa is, therefore, more accurate than the standard Q-learning. We consider only one agent in this experiment. Table 3 and Figure 10 confirm that Sarsa is more efficient than Standard Q-learning in terms of monetary cost.

Table 3 Average monetary cost ($) per iteration for standard Q-learning and Sarsa (phase 1: $0 \le iteration < 20000$, phase 2: $20000 \le iteration < 40000$, phase 3: $40000 \le iteration < 70000$)

	policy 1		
	phase 1	*phase 2*	*phase 3*
Standard QL – cost	0.1518	0.1304	0.1208
Sarsa – cost	0.1519	0.1232	0.1182
	policy 2		
	phase 1	*phase 2*	*phase 3*
Standard QL – cost	0.1522	0.1256	0.1163
Sarsa – cost	0.1525	0.1176	0.1133
	policy 3		
	phase 1	*phase 2*	*phase 3*
Standard QL – cost	0.1344	0.1228	0.1089
Sarsa – cost	0.1269	0.1108	0.1029

Figure 10 Evolution of monetary cost over time (standard Q-learning vs. Sarsa)

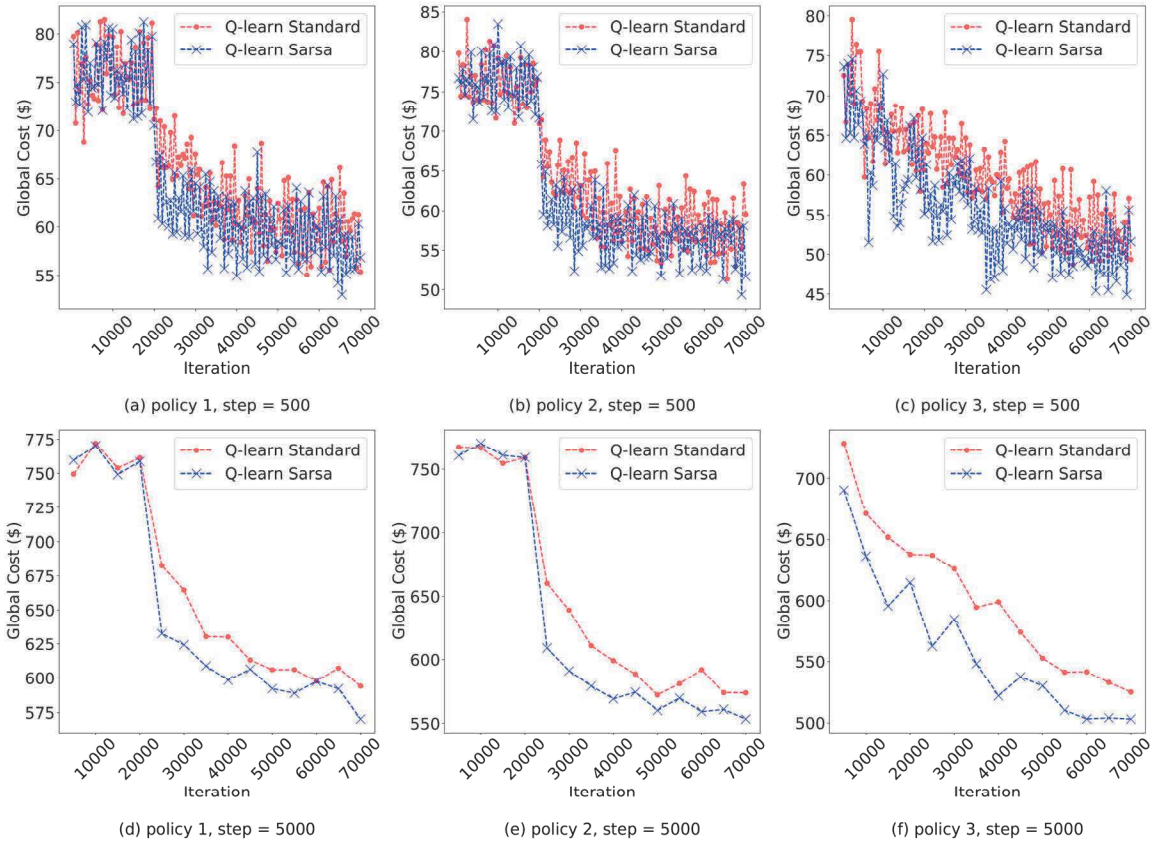
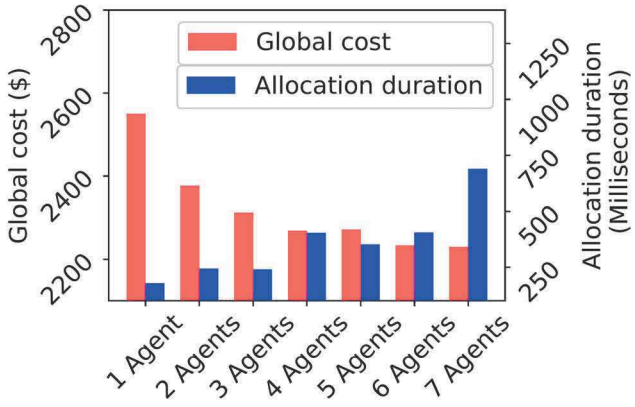


Figure 11 Monetary cost vs. average allocation time



4.5 Discussion

The performed experiments concern three aspects:

- First, reinforcement learning and its applicability to the auto-scaling problem for SQL-like queries. A comparison of our auto-scaling method and an existing method showed the benefit of using the placement-scheduling output. The state definition of the MDP is more precise so the monetary cost is lower.
- Then, the collaboration between agents. We observed that exploiting the shared experience allows agents to learn faster and reduce costs but the allocation time is more important because of the communication between agents.
- Finally, the performance of Sarsa learning. Sarsa variant is more efficient than the standard Q-learning. This can be explained by the fact that Sarsa is based on more precise calculations of $\hat{Q}$. Indeed, Sarsa allows the agent to learn that some of his decisions are random.

5 Related work

The existing methods that deal with auto-scaling are diverse in regards to the decision making approach. Each work has been designed with particular goals and focusing on a target architecture. State-of-the-art methods can be classified in different ways. In this section, we give a brief overview of existing methods and focus on two approaches: (1) threshold based and (2) reinforcement learning. On the one hand, because of its intuitive appearance, the threshold based approach is adopted by the current cloud providers. On the other hand, reinforcement learning is currently experiencing interest from the scientific community and its adoption for different cloud applications and architectures is a promising trend.

5.1 Threshold based approach

Using threshold rules is a well-known approach for auto-scaling in the cloud. The idea is to add new resources when a certain metric exceeds an allocation threshold and to release resources when the metric is below a release threshold. Existing work can be classified into two categories. Some works are based on the observed values of the metric (Han et al., 2012; Hasan et al., 2012) while others apply prediction techniques (Khatua et al., 2010). Han et al. (2012) claim that scaling resources does not always require to add or remove VMs constantly. Modifying VM's capacity (CPU and memory) can be conducted to achieve scaling with fewer costs and less time. They introduced a lightweight algorithm to enable scaling at the level of underlying CPU and memory. The solution uses separate thresholds for the processor and the memory. The considered metrics are based on observation and not predictions. Moreover, communication costs are not considered.

The communication aspect has been considered in some works. Hasan et al. (2012) use link load, jitter, and delay as metrics. They focus on the relation between compute and network. They emphasise the fact that these three domains are often considered separately for scaling. To address this limitation, they proposed a threshold mechanism that combines metrics from computing and network domains. This work also considers only observed values to compute metrics.

Some other works propose to associate the approach of thresholds with a voting process (RightScale). Each VM votes for a scaling action (add or remove resources). The vote is based on one or more rules managed by metrics and thresholds. The scaling action is triggered if the majority of VMs agree. This solution was adopted by Chieu et al. (2011) and Simmons et al. (2011). There are also works that combined the threshold-based approach with other scaling approaches. Ghanbari et al. (2011) propose an elasticity policy using both control theory and threshold based approaches. Unlike the use of thresholds which is more intuitive, the control theory is based on mathematical modeling. RightScale was used in this work as a threshold based management cloud system.

All the works cited above use the observed values to calculate the metrics. Other studies have considered the prediction of future values. We mention, for example, Khatua et al. (2010) who uses time series theory (Mills, 1991) to predict future values. If any of the predicted values exceeds the predefined threshold then an event is triggered.

The intuitive nature of threshold rules attracted cloud providers. However, the choice of metrics to consider and the setting of thresholds in an efficient manner requires human intervention and a deep understanding of the current workload trends which is not easy to achieve. Another approach independent of human intervention interests the scientific community. It is based on reinforcement learning

5.2 Reinforcement learning approach

Reinforcement learning has been adopted for auto-scaling in some cloud work. Existing methods can be differentiated by the scaling mode, the learning algorithm and the technique used to accelerate learning.

Scaling mode in the cloud can be horizontal or vertical. In the horizontal scaling, possible actions are to add or remove resources (Dutreilh et al., 2011). In the vertical scaling, contrariwise, the number of resources is fixed and possible actions consist to adjust their configuration in terms of CPU and memory (Rao et al., 2009, 2011).

The typical reinforcement learning algorithm is standard Q-learning but some works use a variant called Sarsa (Tesauro et al., 2006). The two variants were considered and compared in the experimental section. More details on the difference between standard Q-learning and Sarsa can be found in Alpaydin (2014).

Despite their advantages, reinforcement learning algorithms have certain problems including the large learning time and the size of the state space. Dutreilh et al. (2011) introduce a greedy policy to find a good initialisation of learning values, a convergence speedup technique, and performance model change detection. Rao et al. (2009) propose initially to adopt a global reinforcement learning model for vertical scaling. In this model, the state is described by the amount of CPU/memory of all VMs in the cloud. This approach gives rise to a very large number of states and therefore a lot of time to sufficiently explore the model. Then, Rao et al. (2011) consider that each VM has its own local model and in this case, the state is defined by the amount of CPU/memory of this VM only so the complexity is reduced.

Barrett et al. (2013) use a parallel version of learning but the level of generality is limited to the VM level (not logical resource level), the auto-scaling is independent of placement-scheduling and the specificities of databases querying are not considered in the state description and reward function).

5.3 Discussion

Few works for auto-scaling in the cloud are dedicated to database querying. There are some works that proposed auto-scaling solutions for databases in the cloud but they focused on specific technologies, for example, MongoDB (Huang et al., 2013) or Hadoop (Gandhi et al., 2016). These works made performances (not monetary) metrics in their proposals. Some work focuses on NoSQL databases. For example, TIRAMOLA is an open-source framework to perform auto-scaling of NoSQL clusters according to user-defined policies (Konstantinou et al., 2012; Angelou et al., 2012; Tsoumakos et al., 2013). Decisions on adding or removing workers are modelled as MDPs. There is also existing work on cost-aware horizontal scaling of NoSQL databases (Naskos et al., 2015, 2017, 2018). These proposals are based on MDPs as well and they use probabilistic model checking as the main decision mechanism. In our work we focus on more complex and long-running queries.

The level of granularity in most existing work on auto-scaling is limited to the VM level and queries are seen as atomic entities. In our work, we consider a finer granularity level. Indeed, a VM contains a set of logical resources and each query is decomposed into a job (or stage) graph with dependencies. Each logical resource uses a specific amount of physical resources (CPU, memory, and disk) on a specific machine. A job (or stage) contains a set of parallel tasks. The problem is therefore much more complex if we compare it to web applications.

Finally, the management of intermediate data is generally neglected in the existing work. This is an important feature for data processing applications. Indeed, if the next consumer of intermediate data is not available immediately, then this data might be stored. The use of storage capacity in the cloud has a monetary cost that cannot be ignored. Disks on Amazon S3, for example, are billed according to the size and duration of storage.

6 Conclusion

We addressed in this paper the resource allocation problem for database querying in the cloud. We proposed an auto-scaling method coupled with placement-scheduling. The auto-scaling is based on parallel reinforcement learning and experience sharing. The results show that: (1) considering placement and scheduling plan to describe the MDP is more suitable to long queries than the reinforcement learning methods proposed in previous work, (2) using experience sharing reduces the monetary cost but generates exchanges that increase the allocation duration, (3) the Sarsa variant of Q-learning brings a lower monetary cost compared to the standard Q-learning.

In future work, we will focus more on placement and scheduling problems. So far, we proposed a static method based on estimates (Appendices A and B). The real values may not match those of the estimates at execution time which will generate additional costs. We plan to design a dynamic allocation strategy that detects estimation errors during execution time and change the allocation plan to reduce the impact of these errors.

References

Alpaydin, E. (2014) *Introduction to Machine Learning*, MIT Press.

Angelou, E., Papailiou, N., Konstantinou, I., Tsoumakos, D. and Koziris, N. (2012) 'Automatic scaling of selective sparql joins using the tiramola system', *Proceedings of the 4th International Workshop on Semantic Web Information Management*, ACM, p.1.

Armbrust, M., Xin, R.S., Lian, C. et al. (2015) 'Spark sql: relational data processing in spark', *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM, pp.1383–1394.

Barrett, E., Howley, E. and Duggan, J. (2013) 'Applying reinforcement learning towards automating resource allocation and application scalability in the cloud', *Concurrency and Computation: Practice and Experience*, Vol. 25, No. 12, pp.1656–1674.

Chieu, T.C., Mohindra, A. and Karve, A.A. (2011) 'Scalability and performance of web applications in a compute cloud', *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, IEEE, pp.317–323.

Dean, J. and Ghemawat, S. (2008) 'Mapreduce: simplified data processing on large clusters', *Communications of the ACM*, Vol. 51, No. 1, pp.107–113.

Dean, J. and Ghemawat, S. (2010) 'Mapreduce: a flexible data processing tool', *Communications of the ACM*, Vol. 53, No. 1, pp.72–77.

Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N. and Truck, I. (2011) 'Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow', *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pp.67–74.

Gandhi, A., Thota, S., Dube, P., Kochut, A. and Zhang, L. (2016) 'Autoscaling for hadoop clusters', *Cloud Engineering (IC2E), 2016 IEEE International Conference on*, IEEE, pp.109–118.

Ghanbari, H., Simmons, B., Litoiu, M. and Iszlai, G. (2011) 'Exploring alternative approaches to implement an elasticity policy', *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, IEEE, pp.716–723.

Grounds, M. and Kudenko, D. (2008) 'Parallel reinforcement learning with linear function approximation', *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, Springer, pp.60–74.

Han, R., Guo, L., Ghanem, M.M. and Guo, Y. (2012) 'Lightweight resource scaling for cloud applications', *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, IEEE, pp.644–651.

Hasan, M.Z., Magana, E., Clemm, A., Tucker, L. and Gudreddi, S.L.D. (2012) 'Integrated and autonomic cloud resource scaling', *Network Operations and Management Symposium (NOMS)*, IEEE, pp.1327–1334.

Hromkovič, J. (2013) *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*, Springer Science & Business Media.

Huang, C.-W., Shih, C.-C., Hu, W.-H., Lin, B.-T. and Cheng, C.-W. (2013) 'The improvement of auto-scaling mechanism for distributed database-a case study for mongodb', *Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific*, IEEE, pp.1–3.

Kandi, M.M., Yin, S. and Hameurlain, A. (2018) 'An integer linear-programming based resource allocation method for SQL-like queries in the cloud', *ACM Symposium on Applied Computing (SAC)*, Pau, France.

Khatua, S., Ghosh, A. and Mukherjee, N. (2010) Optimizing the utilization of virtual resources in cloud environment', *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2010 IEEE International Conference on*, IEEE, pp.82–87.

Kllapi, H., Sitaridi, E., Tsangaris, M.M. and Ioannidis, Y. (2011) 'Schedule optimization for data processing flows on the cloud', *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, ACM, pp.289–300.

Konstantinou, I., Angelou, E., Tsoumakos, D., Boumpouka, C., Koziris, N. and Sioutas, S. (2012) 'Tiramola: elastic nosql provisioning through a cloud management platform', *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, pp.725–728.

Kretchmar, R.M. (2002) 'Parallel reinforcement learning', *The 6th World Conference on Systemics, Cybernetics, and Informatics*.

Lawler, E.L. and Wood, D.E. (1966) 'Branch-and-bound methods: a survey', *Operations Research*, Vol. 14, No. 4, pp.699–719.

Mills, T.C. (1991) *Time Series Techniques for Economists*, Cambridge University Press.

Naskos, A., Gounaris, A. and Katsaros, P. (2017) 'Cost-aware horizontal scaling of nosql databases using probabilistic model checking', *Cluster Computing*, Vol. 20, No. 3, pp.2687–2701.

Naskos, A., Gounaris, A. and Konstantinou, I. (2018) 'Elton: a cloud resource scaling-out manager for nosql databases', *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, IEEE, pp.1641–1644.

Naskos, A., Stachtiari, E., Gounaris, A., Katsaros, P., Tsoumakos, D., Konstantinou, I. and Sioutas, S. (2015) 'Dependable horizontal scaling based on probabilistic model checking', *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, IEEE, pp.31–40.

Rao, J., Bu, X., Xu, C.-Z. and Wang, K. (2011) 'A distributed self-learning approach for elastic provisioning of virtualized cloud resources', *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, IEEE, pp.45–54.

Rao, J., Bu, X., Xu, C.-Z., Wang, L. and Yin, G. (2009) 'Vconf: a reinforcement learning approach to virtual machines auto-configuration', *Proceedings of the 6th international conference on Autonomic computing*, ACM, pp.137–146.

Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A. and Curino, C. (2015) 'Apache tez: A unifying framework for modeling and building data processing applications', *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, ACM, pp.1357–1369.

Simmons, B., Ghanbari, H., Litoiu, M. and Iszlai, G. (2011) 'Managing a saas application in the cloud using paas policy sets and a strategy-tree', *Proceedings of the 7th International Conference on Network and Services Management*, International Federation for Information Processing, pp.343–347.

Tesauro, G., Jong, N.K., Das, R. and Bennani, M.N. (2006) 'A hybrid reinforcement learning approach to autonomic resource allocation', *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, IEEE, pp.65–73.

Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S. and Koziris, N. (2013) 'Automated, elastic resource provisioning for nosql clusters using tiramola', *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, IEEE, pp.34–41.

Vavilapalli, V.K., Murthy, A.C., Douglas, C. et al. (2013) 'Apache hadoop yarn: yet another resource negotiator', *Proceedings of the 4th annual Symposium on Cloud Computing*, ACM, p.5.

Watkins, C.J.C.H. (1989) *Learning from Delayed Rewards*, PhD Thesis, King's College, Cambridge.

Yin, S., Hameurlain, A. and Morvan, F. (2018) 'SLA definition for multi-tenant dbms and its impact on query optimization', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 30, No. 11, pp.2213–2226.

Note

1 There are some differences between Hive/Tez and SparkSQL on the technical side but the output of query complication is similar. In both tools, the query is represented by a Directed Acyclic Graph (DAG) that contains stages and parallel tasks.

Appendix A Task placement ILP model

Table A1 shows the sets, parameters, and variables of the placement ILP model. The linear constraints are the following.

The memory amount needed for a task of the stage i must not exceed the available memory amount in the chosen logical resource r:

$$C_m^*(i) * x_{i,m,r} \le C_m(T_y(r)), \forall i \in \mathcal{S}, m \in \mathcal{T}_i, r \in \mathcal{P} \quad \text{(A1)}$$

Each task is placed on one and only one logical resource:

$$\sum_{r \in \mathcal{P}} x_{i,m,r} = 1, \forall i \in \mathcal{S}, m \in \mathcal{T}_i \quad \text{(A2)}$$

In order to ensure intra-stage parallelism, two tasks belonging to the same stage cannot be placed in the same resource:

$$\sum_{m \in \mathcal{T}_i} x_{i,m,r} \le 1, \forall i \in \mathcal{S}, r \in \mathcal{P} \quad \text{(A3)}$$

We recall that $v_{a,b}$ is the maximum amount of data transferred between the tasks placed on the resource a and the task placed on the resource b. This definition satisfies the statement (A4).

$$\begin{aligned} &x_{i,m,r_1} = 1 \; and \; x_{j,r,r_2} = 1 \Rightarrow v_{r_1,r_2} \ge Q_{i,j} \\ &\forall i, j \in \mathcal{S}, m \in \mathcal{T}_i, r \in \mathcal{T}_j, r_1, r_2 \in \mathcal{P}, Q_{i,j} > 0 \end{aligned} \quad \text{(A4)}$$

The statement (A4) can be expressed linearly with:

$$\begin{aligned} &Q_{i,j} * x_{i,m,a} + Q_{i,j} * x_{j,r,b} - v_{a,b} \le Q_{i,j}, \\ &\forall i, j \in \mathcal{S}, m \in \mathcal{T}_i, r \in \mathcal{T}_j, a, b \in \mathcal{P}, Q_{i,j} > 0 \end{aligned} \quad \text{(A5)}$$

We add the constraint (A6) to improve the fair distribution of tasks between resources. Without this constraint, we noticed that tasks are not distributed in a balanced way on the resources. Tasks are more likely to be on the same resource. This reduces communication costs but there is a risk to be in situations where we have some too busy resources (risk of exceeding deadlines) and other resources underexploited. Adding the constraint (A6) makes it possible to take into account the balanced distribution of tasks on the available resources. The compromise between load balancing and communication costs is expressed in the objective function. We introduce the variable $\alpha \in \{0,1....,T\}$. The objective function we present later includes α as a variable to minimise.

$$\sum_{i \in \mathcal{S}} \sum_{m \in \mathcal{T}_i} T_i * x_{i,m,r} + \sum_{t<T} (1 - F_{r,t}) \le \alpha, \forall r \in \mathcal{P} \quad \text{(A6)}$$

The following objective function f takes into account the processor, memory, and network cost. It also takes into account load balancing:

$$\begin{aligned} f = &\sum_{i \in \mathbb{S}} \sum_{m \in \mathbb{T}_i} \sum_{r \in \mathbb{P}} C(r) * T_i * x_{i,m,r} + \\ &\sum_{r_1 \in \mathbb{P}} \sum_{r_2 \in \mathbb{P}} C_{com}(r_1, r_2) * v_{r_1,r_2} + W_{rep} * \alpha \end{aligned}$$

The ILP formulation of the placement problem is:

$$\begin{aligned} &\text{minimise} \quad f \\ &\text{subject to} \quad (A1), (A2), (A3), (A5), (A6) \\ &\qquad x_{i,m,r} \in \{0,1\}, \forall i \in \mathbb{S}, m \in \mathbb{T}_i, r \in \mathbb{P} \\ &\qquad v_{r_1,r_2} \in \{0,1....,UpperBound\}, \forall r_1, r_2 \in \mathbb{P} \\ &\qquad \alpha \in \{0,1....,T\} \end{aligned}$$

In the experimental section, the optimal solution of the placement ILP is found with GLPK software using Branch-and-Bound algorithm (Lawler and Wood, 1966; Hromkovič, 2013).

Table A1 Notation used in the ILP placement and scheduling models

	Sets
$\mathcal{S}$	Set of stages of all submitted queries
$\mathcal{T}_i$	Set of tasks of the stage $i \in \mathcal{S}$
$\mathcal{C}$	Set of resources types, each type is characterised by its memory capacity and monetary cost
$\mathcal{P}$	Set of potential logical resources
$\mathcal{F}$	The set of final stages of the submitted queries
	Parameters
$T_y(r)$	the type of the resource $r \in \mathcal{P}$, $Ty(r) \in \mathcal{C}$
$C_m^*(i)$	The memory amount needed for a task of the stage $i \in \mathcal{S}$
$C_m(c)$	the available memory amount in a type c resource $c \in \mathcal{C}$
$Q_{i,j}$	The amount of data transferred between a task of the stage $i \in \mathcal{S}$ and a task of the stage $j \in \mathcal{S}$
T_i	The local response time of a task from stage $i \in \mathcal{S}$
T	The number of considered future time windows
$F_{r,t}$	Indicate whether the resource a is initially available at the moment t (= 1) or not (= 0), $r \in \mathcal{P}$

Table A1 Notation used in the ILP placement and scheduling models (continued)

	Parameters
$Dist(r_1, r_2)$	The distance between the resource $r_1 \in \mathcal{P}$ and the resource $r_2 \in \mathcal{P}$
W_{proc}	The processor cost weight
W_{mem}	The memory cost weight
W_{com}	Load communication weight
W_{rep}	repartition weight
$C(r)$	The cost of the logical resource $r \in \mathcal{P}$ ($C(r) = W_{proc} + W_{mem} * C_m(T_y(r))$)
$C_{com}(r_1, r_2)$	The cost of communication between the logical resources r_1 and $r_2 \in \mathcal{P}$ ($C_{com}(r_1, r_2) = W_{com} * Dist(r_1, r_2)$)
$S_{i,j}$	indicates whether the stage $i \in \mathcal{S}$ and $j \in \mathcal{S}$ are linked by non-pipeline, i, j , $S_{i,j} \in \{0,1\}$ ("non-pipeline" means that the task j can start from the moment the task i ends completely)
$P_{i,j}$	indicates whether the stage $i \in \mathcal{S}$ and are linked by pipeline, $P_{i,j} \in \{0,1\}$ ("pipeline" means that the task j can start as soon as the task i generates its first output)
$A_{i,m}$	indicates the logical resource in which the task $m \in \mathcal{T}_i$ of the stage $i \in \mathcal{S}$ was placed following the placement phase
D_i	the deadline for the query to which the stage $i \in \mathcal{S}$ belongs
W_i	the penalty weight associated with each run time window after the deadline of the stage $\in \mathcal{S}$
W_s	the weight associated with the storage cost of intermediate results
q_i	the estimated amount of data generated by the stage $i \in \mathcal{S}$ tasks
	Variables
$x_{i,m,r}$	Define whether the task $m \in \mathcal{T}_i$ of stage $i \in \mathcal{S}$ is placed on the resource $r \in \mathcal{P}$ (= 1) or not (= 0), $x_{i,m,r} \in \{0,1\}$
v_{r_1,br_2}	The maximum amount of data transferred between the task placed on the resource $r_1 \in \mathcal{P}$ and the task placed on the resource $r_2 \in \mathcal{P}$
α	fictive variable used in constraint (15), $\alpha \in \{0,1....,T\}$
$y_{i,m,t}$	defines whether the task $m \in \mathcal{T}_i$ of the stage $i \in \mathcal{S}$ started before, at (= 1) or after (= 0) the moment $t \in 0,...,T$, $y_{i,m,t} \in 0,1$
$w_{i,m,t}$	Defines whether the intermediate results of the task $m \in \mathcal{T}_i$ of the stage $i \in \mathcal{S}$ are stored at the moment $t \in \{0,...,T\}$ (= 1) or not (= 0), $w_{i,m,t} \in 0,1$
$\beta_{i,t}$	fictive variable, $i \in \mathcal{S}$, $t \in \{0,...,T\}$

Appendix B Task scheduling ILP model

The output of the placement model is considered as an input to the scheduling ILP model that we present in this section. Table A1 shows the sets, parameters, and variables of the scheduling model. The linear constraints are as follows.

We can deduce from the definition of the family of variables y that:

$$y_{i,m,t} \le y_{i,m,t+1}, \forall i \in \mathcal{S}, m \in \mathcal{T}_i, t < T \quad \text{(B1)}$$

The intermediate results of a task are maintained on the local storage space until all successive tasks begin:

$$y_{i,m_1,t} = 1 \; and \; y_{j,m_2,t} = 0 \Rightarrow w_{i,m_1,t} = 1$$
$$\forall i, j \in \mathcal{S}, m_1 \in \mathcal{T}_i, m_2 \in \mathcal{T}_j, S_{i,j} = 1 \; or \; P_{i,j} = 1 \quad \text{(B2)}$$

This constraint can be expressed linearly as follows:

$$S_{i,j} * y_{i,m_1,t} + S_{i,j} * (1 - y_{j,m_2,t}) - w_{i,m_1,t} \le 1,$$
$$\forall i, j \in \mathcal{S}, m_1 \in \mathcal{T}_i, m_2 \in \mathcal{T}_j, t < T, S_{i,j} > 1 \quad \text{(B3)}$$

$$P_{i,j} * y_{i,m_1,t} + P_{i,j} * (1 - y_{j,m_2,t}) - w_{i,m_1,t} \leq 1,$$
$$\forall i, j \in \mathcal{S}, m_1 \in \mathcal{T}_i, m_2 \in \mathcal{T}_j, t < T, P_{i,j} > 1 \quad \text{(B4)}$$

A resource cannot run more than one task at a time (exclusivity constraint). From the definition of the family of variables y and knowing that a task cannot be interrupted before its end, we can deduce that: $y_{i,m,t} - y_{i,m,t-T_i} = 1$ if the task m of the stage i uses the resource r at moment t; $= 0$ otherwise. The linear formulation of the exclusivity constraint is as follows:

$$\sum_{i \in \mathcal{S}} \sum_{\substack{m \in \mathcal{T}_i \\ A_{i,m} = r}} \sum_{t - T_i \geq 1} (y_{i,m,t} - y_{i,m,t-T_i}) + \sum_{i \in \mathcal{S}} \sum_{\substack{m \in \mathcal{T}_i \\ A_{i,m} = r}} \sum_{t - T_i < 1} y_{i,m,t} \leq F_{r,t}, \forall r \in \mathcal{P}, t < T \quad \text{(B5)}$$

We propose the following formulation for the precedence between tasks constraint. We recall that 'pipeline' means that the task *j* can start as soon as the task *i* generates its first output. 'non-pipeline' means that the task *j* can start from the moment the task *i* ends.

$$y_{j,r,t} - y_{i,m,t-T_i} \leq 1 - S_{i,j},$$
$$\forall i, j \in \mathcal{S}, m \in \mathcal{T}_i, r \in \mathcal{T}_j, t - T_i \geq 1 \quad \text{(B6)}$$

$$y_{j,r,t} \leq 1 - S_{i,j}, \forall i, j \in \mathcal{S}, r \in \mathcal{T}_j, t - T_i < 1 \quad \text{(B7)}$$

$$y_{j,r,t} - y_{i,m,t} \leq 1 - P_{i,j},$$
$$\forall i, j \in \mathcal{S}, m \in \mathcal{T}_i, r \in \mathcal{T}_j, t < T \quad \text{(B8)}$$

The economic costs that influence the scheduling of tasks are penalties and storage of intermediate results. The goal is to find the combination of y and w that minimises this cost. Each query has a deadline specified in SLAs. The accumulation of penalties begins when the execution of the query exceeds the deadline. The objective function to minimise is the following. The first (resp. second) line represents the penalty cost (resp. storage cost):

$$g = \sum_{i \in \mathcal{F}} \sum_{D_i - T_i < t < T} W_i * \max_{r \in \mathcal{T}_i} (1 - y_{i,r,t}) + W_s * \sum_{i \in \mathcal{S}} \sum_{m \in \mathcal{T}_i} \sum_{t < T} q_i * w_{i,m,t} \quad \text{(B9)}$$

This objective function is non-linear. To have a linear form, we introduce the family of variables β such as:

$$1 - y_{i,r,t} \leq \beta_{i,t} \ \forall i \in \mathcal{S}, r \in \mathcal{T}_i, t < T \quad \text{(B10)}$$

The objective function can be expressed linearly as follows:

$$g' = \sum_{i \in \mathcal{F}} \sum_{D_i - T_i < t < T} W_i * \beta_{i,t} + W_s * \sum_{i \in \mathcal{S}} \sum_{m \in \mathcal{T}_i} \sum_{t < T} q_i * w_{i,m,t} \quad \text{(B11)}$$

The ILP formulation for the scheduling problem is:

minimise g'
subject to (B1), (B3), (B4), (B5), (B6), (B7), (B8), (B10)
$y_{i,m,t} \in \{0,1\}, \forall i \in \mathcal{S}, m \in \mathcal{T}_i, t < T$
$w_{i,m,t} \in \{0,1\}, \forall i \in \mathcal{S}, m \in \mathcal{T}_i, t < T$
$\beta_{i,t} \in \{0,1\}, \forall i \in \mathcal{S}, t < T$

The optimal solution of the scheduling ILP is found with GLPK software using Branch-and-Bound algorithm (Lawler and Wood, 1966; Hromkovič, 2013).